ON VARIOUS NOTIONS OF PARALLELISM IN P SYSTEMS

OSCAR H. IBARRA*

Department of Computer Science, University of California, Santa Barbara, CA 93106, USA

HSU-CHUN YEN†

Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan 106, R.O.C.

ZHE DANG‡

*School of Electrical Engineering and Computer Science, Washington State University, Pullman,
WA 99164, USA*

Received 23 January 2005

Accepted 2 April 2005

Communicated by C. S. Calude

ABSTRACT

*Corresponding author. The research of Oscar H. Ibarra was supported in part by NSF Grants IIS-0101134, CCR-0208595, and CCF-0430945.

†The research of Hsu-Chun Yen was supported in part by NSC Grant 93-2213-E-002-003, Taiwan.

‡The research of Zhe Dang was supported in part by NSF Grant CCF-0430531.

We consider the following definition (different from the standard definition in the literature) of “maximal parallelism” in the application of evolution rules in a P system G : Let $R = \{r_1, \dots, r_k\}$ be the set of (distinct) rules in the system. G operates in maximally parallel mode if at each step of the computation, a maximal subset of R is applied, and at most one instance of any rule is used at every step (thus at most k rules are applicable at any step). We refer to this system as a maximally parallel system. We look at the computing power of P systems under three semantics of parallelism. For a positive integer $n \leq k$, define:

n -Max-Parallel: At each step, nondeterministically select a maximal subset of at most n rules in R to apply (this implies that no larger subset is applicable).

$\leq n$ -Parallel: At each step, nondeterministically select any subset of at most n rules in R to apply.

n -Parallel: At each step, nondeterministically select any subset of exactly n rules in R to apply.

In all three cases, if any rule in the subset selected is not applicable, then the whole subset is not applicable. When $n = 1$, the three semantics reduce to the **Sequential** mode.

We focus on two popular models of P systems: multi-membrane catalytic systems and communicating P systems. We show that for these systems, **n -Max-Parallel** mode is strictly more powerful than any of the following three modes: **Sequential**, **$\leq n$ -Parallel**, or **n -Parallel**. For example, it follows from the result in [9] that a maximally parallel communicating P system is universal for $n = 2$. However, under the three limited modes of parallelism, the system is equivalent to a vector addition system, which is known to only define a recursive set. These generalize and refine the results for the case of 1-membrane systems recently reported in [3]. Some of the present results are rather surprising. For example, we show that a **Sequential** 1-membrane communicating P system can only generate a semilinear set, whereas with k membranes, it is equivalent to a vector addition system for any $k \geq 2$ (thus the hierarchy collapses at 2 membranes - a rare collapsing result for nonuniversal P systems). We also give another proof (using vector addition systems) of the known result [8] that a 1-membrane catalytic system with only 3 catalysts and (non-prioritized) catalytic rules operating under **3-Max-Parallel** mode can simulate any 2-counter machine M . Unlike in [8], our catalytic system needs only a *fixed* number of noncatalysts, independent of M .

A simple cooperative system (SCO) is a P system where the only rules allowed are of the form $a \rightarrow v$ or of the form $aa \rightarrow v$, where a is a symbol and v is a (possibly null) string of symbols not containing a . We show that a **9-Max-Parallel** 1-membrane SCO is universal.

1. Introduction

There has been a flurry of research activities in the area of membrane computing (a branch of molecular computing) initiated five years ago by Gheorghe Paun [15]. Membrane computing identifies an unconventional computing model, namely a P system, from natural phenomena of cell evolutions and chemical reactions. Due to the built-in nature of maximal parallelism inherent in the model, P systems have a great potential for implementing massively concurrent systems in an efficient way that would allow us to solve currently intractable problems (in much the same way as the promise of quantum and DNA computing) once future bio-technology (or silicon-technology) gives way to a practical bio-realization (or chip-realization).

The Institute for Scientific Information (ISI) has recently selected membrane computing as a fast “Emerging Research Front” in Computer Science (<http://esitopics.com/erf/october2003.html>). A P system is a computing model, which abstracts from the way the living cells process chemical compounds in their compart-

mental structure. Thus, regions defined by a membrane structure contain objects that evolve according to given rules. The objects can be described by symbols or by strings of symbols, in such a way that multisets of objects are placed in regions of the membrane structure. The membranes themselves are organized as a Venn diagram or a tree structure where one membrane may contain other membranes. By using the rules in a nondeterministic, maximally parallel manner, transitions between the system configurations can be obtained. A sequence of transitions shows how the system is evolving. Various ways of controlling the transfer of objects from a region to another and applying the rules, as well as possibilities to dissolve, divide or create membranes have been studied. P systems were introduced with the goal to abstract a new computing model from the structure and the functioning of the living cell (as a branch of the general effort of Natural Computing – to explore new models, ideas, paradigms from the way nature computes). Membrane computing has been quite successful: many models have been introduced, most of them Turing complete and/or able to solve computationally intractable problems (NP-complete, PSPACE-complete) in a feasible time (polynomial), by trading space for time. (See the P system website at <http://psystems.disco.unimib.it> for a large collection of papers in the area, and in particular the monograph [16].)

As already mentioned above, in the standard semantics of P systems [15, 16, 17], each evolution step of a system G is a result of applying all the rules in G in a maximally parallel manner. More precisely, starting from the initial configuration, w , the system goes through a sequence of configurations, where each configuration is derived from the directly preceding configuration in one step by the application of a multi-set of rules, which are chosen nondeterministically. For example, a catalytic rule $Ca \rightarrow Cv$ in membrane q is applicable if there is a catalyst C and an object (symbol) a in the preceding configuration in membrane q . The result of applying this rule is the evolution of v from a . If there is another occurrence of C and another occurrence of a , then the same rule or another rule with Ca on the left hand side can be applied. Thus, in general, the number of times a particular rule is applied at anyone step can be unbounded. We require that the application of the rules is maximal: all objects, from all membranes, which *can be* the subject of local evolution rules *have to* evolve simultaneously. Configuration z is reachable (from the starting configuration) if it appears in some execution sequence; z is halting if no rule is applicable on z .

In this paper, we study a different definition of maximal parallelism. Let G be a P system and $R = \{r_1, \dots, r_k\}$ be the set of (distinct) rules in all the membranes. (Note that r_i uniquely specifies the membrane the rule belongs to.) We say that G operates in maximally parallel mode if at each step of the computation, a maximal subset of R is applied, and at most one instance of any rule is used at every step (thus at most k rules are applicable at any step). For example, if r_i is a catalytic rule $Ca \rightarrow Cv$ in membrane q and the current configuration has two C 's and three a 's in membrane q , then only one a can evolve into v . Of course, if there is another rule r_j , $Ca \rightarrow Cv'$, in membrane q , then the other a also evolves into v' . Throughout the paper, we will use this definition of maximal parallelism. Here, we look at the

computing power of P systems under three semantics of parallelism. For a positive integer $n \leq k$, define:

n -Max-Parallel: At each step, nondeterministically select a maximal subset of at most n rules in R to apply (this implies that no larger subset is applicable).

$\leq n$ -Parallel: At each step, nondeterministically select any subset of at most n rules in R to apply.

n -Parallel: At each step, nondeterministically select any subset of exactly n rules in R to apply.

In all three cases, if any rule in the subset selected is not applicable, then the whole subset is not applicable. When $n = 1$, the three semantics reduce to the **Sequential** mode.

In the next four sections, we investigate the computing power of two popular models of P systems with respect to the above semantics of parallelism – the catalytic P systems and the communicating P systems.

We should mention some related work on P systems operating in sequential and limited parallel modes. Sequential variants of P systems have been studied, in a different framework, in [7]. There, generalized P systems (GP-systems) were considered and were shown to be able to simulate graph controlled grammars. Our notion of limited parallelism seems to correspond to “cooperation modes” in cooperating distributed grammar systems, investigated in [2].

2. Multi-Membrane Catalytic Systems

2.1. Maximally Parallel CS

First we recall the definition of a catalytic system (CS). The membranes (regions) are organized in a hierarchical (tree) structure and are labeled $1, 2, \dots, m$ for some m , with the outermost membrane (the skin membrane) labeled 1. At the start of the computation, there is a distribution of *catalysts* and *noncatalysts* in the membranes (the distribution represents the initial configuration of the system). Each membrane may contain a finite set of catalytic rules of the form $Ca \rightarrow Cv$, where C is a catalyst, a is a noncatalyst, and v is a (possibly null) string of noncatalysts. When this rule is applied, the catalyst remains in the membrane the rule is in, symbol a is deleted from the membrane, and the symbols comprising v (if nonnull) are transported to other membranes in the following manner. Each symbol b in v has a designation or target, i.e., it is written b_x , where x can be *here*, *out*, or *in_j*. The designation *here* means that the object b remains in the membrane containing it (we usually omit this target, when it is understood). The designation *out* means that the object is transported to the membrane directly enclosing the membrane that contains the object; however, we do not allow any object to be transported out of the skin membrane. The designation *in_j* means that the object is moved into a membrane, labeled j , that is directly enclosed by the membrane that contains the object.

It is important to note that our definition of catalytic system is different from

what is usually called catalytic system in the literature. Here, we do not allow rules without catalysts, i.e., rules of the form $a \rightarrow v$. Thus our systems use only purely catalytic rules.

Suppose that S is a CS with m membranes. Let $\{a_1, \dots, a_n\}$ be the set of noncatalyst symbols (objects) that can occur in the configurations of S . Let $w = (w_1, \dots, w_m)$ be the initial configuration, where w_i represents the catalysts and noncatalysts in membrane i . (Note that w_i can be null.) Each reachable configuration of S is an nm -tuple $(v_1, \dots, v_m)$, where v_i is an n -tuple representing the multiplicities of the symbols $a_1, \dots, a_n$ in membrane i . Note that we do not include the catalysts in considering the configuration as they are not changed (i.e., they remain in the membranes containing them, and their numbers remain the same during the computation). Hence the set of all reachable configurations of S , denoted by $R(S)$ is a subset of $\mathbf{N}^{mn}$. The set of all halting reachable configurations is denoted by $R_h(S)$.

2.2. Sequential CS

In a sequential multi-membrane CS, each step of the computation consists of an application of a single nondeterministically chosen rule, i.e., the membrane and rule within the membrane to apply are chosen nondeterministically. We show below that sequential multi-membrane CS's define exactly the semilinear sets.

We need the definition of a vector addition system. An n -dimensional *vector addition system* (VAS) is a pair $G = \langle x, W \rangle$, where $x \in \mathbf{N}^n$ is called the *start point* (or *start vector*) and W is a finite set of transition vectors in $\mathbf{Z}^n$, where $\mathbf{Z}$ is the set of all integers (positive, negative, zero). Throughout this paper, for a $w \in \mathbf{Z}^n$ we write $w \geq 0$ to mean that w has only nonnegative components (i.e., $w \in \mathbf{N}^n$). The *reachability set* of the VAS $\langle x, W \rangle$ is the set $R(G) = \{z \mid \text{for some } j, z = x + v_1 + \dots + v_j, \text{ where, for all } 1 \leq i \leq j, \text{ each } v_i \in W \text{ and } x + v_1 + \dots + v_i \geq 0\}$. Note that $R(G)$ is the smallest set satisfying the following two properties: (1) $x \in R(G)$, and (2) whenever $z \in R(G)$, $v \in W$, and $z + v \in \mathbf{N}^n$, then $z + v \in R(G)$. The *halting reachability set* $R_h(G) = \{z \mid z \in R(G), z + v \not\geq 0 \text{ for every } v \text{ in } W\}$.

An n -dimensional *vector addition system with states* (VASS) is a VAS $\langle x, W \rangle$ together with a finite set T of transitions of the form $p \rightarrow (q, v)$, where q and p are states and v is in W . The meaning is that such a transition can be applied at point y in state p and yields the point $y + v$ in state q , provided that $y + v \geq 0$. The VASS is specified by $G = \langle x, W, T, p_0 \rangle$, where p_0 is the starting state. Assuming that the set of states of G is $\{p_0, \dots, p_k\}$ (for some $k \geq 0$), the *reachability set* of VASS G is $R(G) = \{(i, w) \in \mathbf{N}^{n+1} \mid 1 \leq i \leq k, (p_i, w) \text{ is reachable from } (p_0, x)\}$. The *halting reachability set* $R_h(G) = \{(i, w) \mid (i, w) \in R(G) \text{ and no transition is applicable in } (p_i, w)\}$.

The *reachability problem* for a VASS (respectively, VAS) G is to determine, given a vector y , whether y is in $R(G)$. The *equivalence problem* is to determine given two VASS (respectively, VAS) G and G' , whether $R(G) = R(G')$. Similarly, one can define the reachability problem and the equivalence problem for halting configurations.

The following summarizes the known results concerning VAS and VASS [20, 10, 1, 11, 14]:

Theorem 1

1. Let G be an n -dimensional VASS. We can effectively construct an $(n + 3)$ -dimensional VAS G' that simulates G .
2. If G is a 2-dimensional VASS, then $R(G)$ is an effectively computable semilinear set.
3. There is a 3-dimensional VASS G such that $R(G)$ is not semilinear.
4. If G is a 5-dimensional VAS, then $R(G)$ is an effectively computable semilinear set.
5. There is a 6-dimensional VAS G such that $R(G)$ is not semilinear.
6. The reachability problem for VASS (and hence also for VAS) is decidable.
7. The equivalence problem for VAS (and hence also for VASS) is undecidable.

Clearly, it follows from part 6 of the theorem above that the halting reachability problem for VASS (respectively, VAS) is decidable.

A *communication-free VAS* is a VAS where in every transition, at most one component is negative, and if negative, its value is -1. Communication-free VAS's are equivalent to communication-free Petri nets, which are also equivalent to commutative context-free grammars [4, 12]. It is known that they have effectively computable semilinear reachability sets [4].

Our first result shows that a sequential CS is weaker than a maximally parallel CS.

Theorem 2 *Every sequential multi-membrane CS S can be simulated by a communication-free VAS G , and vice versa.*

Proof. Let S be an m -membrane CS with noncatalysts $a_1, \dots, a_n$. Suppose that the start configuration of $w = (w_1, \dots, w_m)$ has k catalysts $C_1, \dots, C_k$. We may assume, without loss of generality by adding new catalysts and rules if necessary, that each C_i occurs at most once in w_i ($1 \leq i \leq m$). Number all the rules in S by $1, \dots, s$. Note that the rule number uniquely determines the membrane where the rule is applicable.

We first transform S to a new system S' by modifying the rules and the initial configuration w . S' will now have catalysts $C_1, \dots, C_k, Q_1, \dots, Q_s$ and noncatalysts $a_1, \dots, a_n, d_1, \dots, d_s$. The component w_q of the initial configuration in membrane q will now be w_q plus each Q_h for which rule number h is in membrane q . The rules of S' are defined as follows:

Case 1: Suppose that $C_j a_i \rightarrow C_j v$ is a rule in membrane q of S , and a_i does not appear in v with designation (target) *here*. Then this rule is in membrane q of S' .

Case 2: Suppose that $C_j a_i \rightarrow C_j a_i^t v$ is rule number r and $t \geq 1$. Suppose that this

rule is in membrane q , with the target of each a_i in a_i^t being *here*, and v does not contain any a_i with target *here*. Then the following rules are in membrane q of S' : $C_j a_i \rightarrow C_j d_r^t v$ and $Q_r d_r \rightarrow Q_r a_i$. In the above rules, the target for a_i and each d_r in the right-hand side of the rules is *here*.

Clearly, S' simulates S , and S' has the property that in each rule $Xb \rightarrow Xv$ (where X is a catalyst, b is a noncatalyst, and v a string of noncatalysts), v does not contain a b with target *here*. It is now obvious that each rule $Xb \rightarrow Xv$ in S' can be transformed to a VAS transition rule of $mn + s$ components, where the component of the transition corresponding to noncatalyst b is -1, and the other components (corresponding to the target designations in v) are nonnegative. Thus, the VAS is communication free.

Conversely, let G be a communication-free VAS. We construct a sequential 1-membrane CS S which has one catalyst C , noncatalysts $\#, a_1, \dots, a_k$, and starting configuration $C\#w$, where w corresponds to the starting vector of G . Suppose that $(j_1, \dots, j_{m-1}, j_m, j_{m+1}, \dots, j_k)$ is a transition in G .

Case 1: $j_m = -1$ and all other j_i 's are nonnegative. Then the following rule is in S : $Ca_m \rightarrow Ca_1^{j_1} \dots a_{m-1}^{j_{m-1}} a_{m+1}^{j_{m+1}} \dots a_k^{j_k}$.

Case 2: All the j_i 's are nonnegative. Then the following rule is in S : $C\# \rightarrow C\#a_1^{j_1} \dots a_k^{j_k}$.

Clearly, S simulates G . In fact, $R(G) = R(S) \times \{1\}$. □

Corollary 1

1. If S is a sequential multi-membrane CS, then $R(S)$ and $R_h(S)$ are effectively computable semilinear sets.
2. The reachability problem (whether a given configuration is reachable) for sequential multi-membrane CS's is NP-complete.

Proof. For part 1, let G be the communication-free VAS constructed in the first part of the proof of Theorem 2. Then $R(G) \subseteq \mathbf{N}^{mn+s}$ is semilinear (since a communication-free VAS has a semilinear reachability set). Now the first mn components of each vector in $R(G)$ correspond to the multiplicities of $a_1, \dots, a_n$ for each of the m membranes, and the remaining s components correspond to $d_1, \dots, d_s$. Clearly, from the proof of the theorem, $R(S) = \text{proj}_{mn}(R(G) \cap (\mathbf{N}^{mn} \times \{0\}^s))$, where proj_{mn} is the projection of the tuples on the first mn coordinates. Since $\mathbf{N}^{mn} \times \{0\}^s$ is semilinear and semilinear sets are closed under intersection and projection, it follows that $R(S)$ is semilinear.

Now consider $R_h(S)$. A noncatalyst b is useful for membrane q if there is a rule in membrane q with Cb on the left hand side for some catalyst C . Without loss of generality (by simple relabeling), assume that the last t positions in the $(mn + s)$ -tuples of transition rules correspond to useful symbols. Note that the same symbol b can not be useful for more than one membrane because in the configurations, we distinguish a symbol occurring in membrane q from the same symbol occurring in another membrane p . Clearly, a reachable configuration in $R(S)$ is halting if coordinates $(mn - t + 1), \dots, mn$ (corresponding to the useful symbols) are zero. Hence $R_h(S) = R(S) \cap (\mathbf{N}^{mn-t} \times \{0\}^t)$, which is semilinear.

Part 2 follows from the NP-completeness of the reachability problem for communication-free Petri nets (which are equivalent to commutative context-free grammars) [12, 4]. $\square$

Since a communication-free VAS can be simulated by a sequential 1-membrane CS (from conversely in the proof of Theorem 2), we have:

Corollary 2 *The following are equivalent: communication-free VAS, sequential multi-membrane CS, sequential 1-membrane CS.*

2.3. CS Under Limited Parallelism

Here we look at the computing power of the multi-membrane CS under three semantics of parallelism (namely, n -**Max-Parallelism**, $\leq n$ -**Parallelism**, and n -**Parallelism**) defined in Section 1. We can show the following:

Theorem 3 *For $n = 3$, a 1-membrane CS operating under the n -Max-Parallel mode can define any recursively enumerable set. For any n , a multi-membrane CS operating under $\leq n$ -Parallel mode or n -Parallel mode can be simulated by a VASS.*

Proof. The first part follows from a recent result in [8], where it was shown that a 1-membrane CS with three catalysts (even when each catalyst appears exactly once in the initial configuration) are already sufficient for universality. It remains an interesting open question as to whether the three catalysts can be reduced to two. One catalyst is not enough as was shown in [13].

For the case of $\leq n$ -Parallel mode, given a multi-membrane CS S , we construct a VASS G to simulate S as follows. G has a unique state p such that G is always in this state at the beginning of every simulation of a parallel step of S . Using several new states, G nondeterministically picks $k \leq n$ rules of the form $Ca_i \rightarrow Cv_i$ (k itself is also nondeterministically chosen) and apply each rule sequentially by subtracting 1 from the coordinate corresponding to a_i in an n -tuple transition rule. G then sequentially updates the coordinates as given in the right-hand sides of the rule $Ca_i \rightarrow Cv_i$ (G needs additional new states to do this) and enters state p . Clearly, a tuple is reachable in S if and only if it is reachable in G in state p . We omit the details. The case of n -Parallel mode follows directly in that G chooses k to be exactly n . $\square$

2.4. 3-Max-Parallel 1-Membrane CS

As noted above, it is known that a 3-Max-Parallel 1-membrane CS is universal [8] in that it can simulate any 2-counter machine M . We show in this section that the universality result of [8] can be obtained in terms of communication-free Petri nets operating under the 3-Max-Parallel mode. Later we improve this result by showing that, in fact, the 1-membrane CS need no more than k noncatalysts for some fixed k , independent of M .

With their computational capabilities equivalent to that of VAS (and VASS), *Petri nets* are a very popular mathematical model for describing concurrent systems and their behaviors, partly because of their appealing pictorial representa-

tions which, in many cases, make the modelling as well as the reasoning easier in comparison with their equivalent counterparts.

A *Petri net* (PN, for short) is a 3-tuple (P, T, φ) , where P is a finite set of *places*, T is a finite set of *transitions*, and φ is a *flow function* $\varphi : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$. A *marking* is a mapping $\mu : P \rightarrow \mathbb{N}$. A transition $t \in T$ is *enabled* at a marking μ iff for every $p \in P$, $\varphi(p, t) \leq \mu(p)$. A transition t may *fire* at a marking μ to yield marking μ' with $\mu'(p) = \mu(p) - \varphi(p, t) + \varphi(t, p)$ for all $p \in P$, if t is enabled at μ . A place p is said to be an *input place* (resp., *output place*) of transition t if $\varphi(p, t) > 0$ (resp. $\varphi(t, p) > 0$). By establishing an ordering on the elements of P (i.e., $P = \{p_1, \dots, p_n\}$), a marking μ can be regarded as an n -dimensional vector $(m_1, \dots, m_n)$ in $\mathbb{N}^n$ such that $\mu(p_i) = m_i, 1 \leq i \leq n$. For ease of expression, we also write μ as a string $(p_1)^{m_1} \dots (p_n)^{m_n}$ in our subsequent discussion. The *reachability set* (resp. *halting reachability set*) of a PN G with respect to initial marking μ_0 , written as $R(G, \mu_0)$ (resp., $R_h(G, \mu_0)$) is defined similarly as that of VAS.

Pictorially, places (resp., transitions) are drawn as circles $\bigcirc$ (resp., boxes $\square$) in Petri net graphs. For a $t \in T$ and $p \in P$, the arc from p to t (resp., from t to p) in the PN graph is annotated by $\varphi(p, t)$ (resp., $\varphi(t, p)$). A marking can be thought of as assigning the so-called *tokens* to the places of a PN. Each token is represented by a small dot $\bullet$. The reader is referred to [18] for more about PNs. A PN (P, T, φ) is called *communication-free* if $\forall t \in T, |\{p \mid \varphi(p, t) > 0\}| = 1$, (i.e., every transition has exactly one input place), and $\forall p \in P, t \in T, \varphi(p, t) \leq 1$ (i.e., every arc going from a place to a transition has weight 1). It is easy to observe that communication-free PN and communication-free VAS (defined in Section 2.2) are computationally equivalent.

Consider an n -place communication-free PN $G = (P, T, \varphi)$ with its set of transitions T partitioned into three disjoint sets W_1, W_2 and W_3 . For ease of expression, we write such a PN as $(P, (W_1, W_2, W_3), \varphi)$. A marking $\mu \in \mathbb{N}^n$ is said to follow $\mu' \in \mathbb{N}^n$ under the **3-Max-Parallel** mode if there exist $t_i \in (W_i \cup \{0\}), 1 \leq i \leq 3$, such that $\forall p \in P$

$$(1) \mu(p) - \sum_{t_i (t_i \neq 0)} \varphi(p, t_i) \geq 0,$$

$$(2) \text{ if } t_i = 0, \text{ then there is no other } t'_i \in W_i \text{ such that (1) holds if } t_i \text{ is replaced by } t'_i, \text{ and}$$

$$(3) \mu'(p) = \mu + \sum_{t_i (t_i \neq 0)} (\varphi(t_i, p) - \varphi(p, t_i)).$$

In this case, we write $\mu \xrightarrow{(t_1, t_2, t_3)} \mu'$. Intuitively, the semantics of **3-Max-Parallelism** requires that at each step, nondeterministically a maximal set of at most 3 transitions be applied simultaneously to yield the next marking; however, from each set $W_i, 1 \leq i \leq 3$, at most one transition can be chosen. Throughout the rest of this paper, the above PNs are referred to as communication-free PNs under the **3-Max-Parallel** mode.

Theorem 4 *Let M be a 2-counter machine with two counters. There exists a 1-membrane 3-Max-Parallel CS S with catalysts C_1, C_2, C_3 and n noncatalysts with*

a designated noncatalyst symbol a_l such that M accepts on initial counter values m and 0, respectively, iff

1. (generator:) $m \in \{\#_{a_l}(y) \mid y \in R_h(S)\};$
2. (acceptor) if S starts with initial configuration $(a_l)^m y$, for some y not containing a_l , then $R_h(S) \neq \emptyset$.

Here n is bounded by a function of the number of states of M .

Proof. We first argue that 1-membrane 3-catalyst 3-**Max-Parallel** CS and communication-free PN under the 3-**Max-Parallel** mode are computationally equivalent. Let $G = (P, (W_1, W_2, W_3), \varphi)$ be a communication-free PN, where $P = \{s_1, \dots, s_n\}$. By assigning for each $1 \leq i \leq 3$, a catalyst C_i for the set of transitions W_i , defining a distinct noncatalyst symbol a_j for each place $s_j, 1 \leq j \leq n$, and converting each transition t in W_i to a rule of the form $C_i a_j \rightarrow C_i v$, where a_j is a noncatalyst and v is a (possibly null) string of noncatalysts such that $\varphi(s_j, t) = 1$ and $\#_{a_h}(v) = \varphi(t, s_h), 1 \leq h \leq n$, an equivalent CS can be constructed. The converse, i.e., converting a 1-membrane 3-catalyst 3-**Max-Parallel** CS to an equivalent communication-free PN under the 3-**Max-Parallel** mode, is similar.

In what follow, we establish the universality result for CS in terms of PNs. We feel that with their appealing graphical representations, PN formulations make the somewhat involved proof easier to understand. To this end, we show that there exist an n -place PN $G = (P, (W_1, W_2, W_3), \varphi)$ with initial marking x under the 3-**Max-Parallel** mode and a place l such that M accepts on initial counter values $C_1 = m$ and $C_2 = 0$ iff (1) (generator:) $m \in \{v(l) \mid v \in R_h(G, x)\};$ (2) (acceptor:) from initial marking x with $x(l) = m$, $R_h(G, x) \neq \emptyset$, i.e., G has a halting computation.

Let the set of states of M be $\{p_1, \dots, p_k\}$, where p_1 the initial state and p_k the final state. W.l.o.g., we assume that whenever M enters an accepting state, both counters are empty. Furthermore, M runs forever if it does not accept. In what follows, we show how to construct an n -place communication-free PN $G = (P, (W_1, W_2, W_3), \varphi)$ with initial marking x meeting Statement (1) of the theorem. To better explain how PN G functions, we divide the n places into the following:

- l : the designated place,
- c_1 and c_2 : keep track of the values of the two counters C_1 and C_2 , respectively; c'_1 and c'_2 : serve as temporary holders for decrementing C_1 and C_2 , respectively, or testing C_1 and C_2 for zero, respectively,
- y and y' : serve for the purpose of generating an arbitrary (but equal) number of tokens in c_1 and l for the simulation to start,
- $p_1, \dots, p_k$: simulate the k states of M in such a way that at any instant, only one of $p_1, \dots, p_k$ has tokens (indicating the current state of M). Furthermore, if from state p_i the next transition to simulate is of type 'increment', then p_i has two tokens; otherwise, p_i has one.

Fig. 1. A PN under the **3-Max-Parallel** mode to simulate a 2-counter machine.

- $s_{(i,j),d}^h, 1 \leq h \leq 5$ ($d = 1$ or 2) serve for simulating transition $q_i \xrightarrow{C_d := C_d - 1} q_j$,
 $t_{(i,j),d}^1$ and $t_{(i,j),d}^2$ ($d = 1$ or 2) for simulating transition $q_i \xrightarrow{C_d = 0} q_j$,
- f : records the occurrence of an illegitimate simulation step during the course of the simulation.

For ease of explanation, transitions associated with W_1 , W_2 and W_3 are drawn as *solid*, *dashed* and *dotted* boxes, respectively, in the PN graph.

The initial marking x of G is $x(y) = 1$, while the remaining places are empty. By using the set of transitions shown in Figure 1(a), an arbitrary value m can be generated in both c_1 and l through firing transitions h_1 and h_2 alternatively m times each. Place f serves as a ‘trap’ in the sense that if it ever holds a token, then the PN computation will become nonterminating. Such a mechanism enforces certain transitions to be taken at the proper time if the trap is to be kept empty at all times. With the installation of α_1 and α_2 , the output transition r_1 of place c_1 (Figure 1(b)) cannot be applied until a token is deposited in place p_1 , signaling the beginning of the next phase; otherwise, the token in either y or y' is forced to move to f under the **3-Max-Parallel** mode since h_1 , h_2 and r_1 are of the same type W_1 . The simulation proceeds to the next phase when a token is moved from place y to place p_1 through the firing of transition h_3 in Figure 1(a).

We are now in a position to see how the three types of transitions of M can be faithfully simulated in G .

- $p_i \xrightarrow{C_d:=C_d-1} p_j, d = 1 \text{ or } 2.$

We consider the case $d = 2$; $d = 1$ is symmetric. The PN structure shown in Figure 1(c), together with Figure 1(b), serves for the purpose of simulating such a transition. (Note that for $d = 1$ the structure of the PN is identical to Figure 1(c) except that the types to which transitions $s^1_{(i,j),d}, s^2_{(i,j),d}, s^3_{(i,j),d}, s^4_{(i,j),d}$ and $s^5_{(i,j),d}$ belong are different.) If the next transition to be simulated from p_j is ‘increment’, then $\varphi(w_6, p_j) = 2$.

Suppose the simulation begins with marking $(c_1)^a(c_2)^b(p_i)$ denoting a configuration of M in which the values of counters C_1 and C_2 are a and b , respectively, and the current state is p_i . First consider the case when $a > 0$ and $b > 0$. In this marking, only r_1, r_2, γ_1 and w_1 are enabled. Since r_1 and w_1 belong to the same set W_1 , only one of them is applicable. Hence, there are two feasible steps:

$$(c_1)^a(c_2)^b(p_i) \xrightarrow{(r_1, r_2, \gamma_1)} (c_1)^{a-1}(c'_1)(c_2)^{b-1}(c'_2)(f)$$

or

$$(c_1)^a(c_2)^b(p_i) \xrightarrow{(w_1, r_2, 0)} (c_1)^a(c_2)^{b-1}(c'_2)(s^1_{(i,j),2})(s^2_{(i,j),2})$$

Clearly the former gives rise to a marking in which the trap f becomes nonempty, rendering the simulation nonterminating. For the latter, it is easy to see that the following sequence of steps is legitimate:

$$(c_1)^a(c_2)^b(p_i) \xrightarrow{(w_1, r_2, 0)} (c_1)^a(c_2)^{b-1}(c'_2)(s^1_{(i,j),2})(s^2_{(i,j),2}) \xrightarrow{(w_2, r'_2, 0)} (c_1)^a(c_2)^{b-1}(s^2_{(i,j),2}) \\ (s^3_{(i,j),2}) \xrightarrow{(w_4, w_3, 0)} (c_1)^a(c_2)^{b-1}(s^4_{(i,j),2})(s^5_{(i,j),2}) \xrightarrow{(w_5, w_6, 0)} (c_1)^a(c_2)^{b-1}(p_j)$$

A careful examination of the above sequence reveals that although r_1 or r_2 might be enabled in each of the intermediate markings during Steps 2-4, executing r_1 or r_2 in Steps 2-4 forces one of $\beta_1, \beta_2, \gamma_2, \dots, \gamma_5$ to take place under the **3-Max-Parallel** mode, which in turn move a token to f . To see this, consider Step 3 from marking $(c_1)^a(c_2)^{b-1}(s^2_{(i,j),2})(s^3_{(i,j),2})$, for instance. If, instead of firing $(w_4, w_3, 0)$, $(w_4, r_2, 0)$ was taken moving a token to c'_2 , a token to $s^5_{(i,j),2}$, while leaving the token in $s^2_{(i,j),2}$ intact. Then since r'_2, w_3 and w_6 are all of type W_2 , only one can be applied at the next step, forcing either γ_5 or β_2 to be applied, which moves a token to f .

In view of the above, the sequence listed above is the only way that the computation of G has a potential to terminate, and such a computation corresponds to a faithful simulation of $q_i \xrightarrow{C_2:=C_2-1} q_j$. The case when $a = 0$ and $b > 0$ is similar. Now consider the case when $C_2 = 0$ (i.e., $b=0$). Then again w_1 must be involved in the first step (otherwise, γ_1 would have been applied, giving rise to a token in place f) which inhibits r_1 from being executed. So we have the following sequence:

$$(c_1)^a(p_i) \xrightarrow{(w_1, 0, 0)} (c_1)^a(s^1_{(i,j),2})(s^2_{(i,j),2}) \xrightarrow{(w_2, w_3, 0)} (c_1)^a(s^3_{(i,j),2})(s^4_{(i,j),2}), \text{ enabling both } w_4 \text{ and } w_5. \text{ However, only one of them can be executed due to the fact that both are of type } W_2. \text{ Hence, one of } \gamma_3 \text{ and } \gamma_4 \text{ is forced to take place, depositing a token in } f. \text{ This in turn implies that simulating } q_i \xrightarrow{C_2:=C_2-1} q_j \text{ while } C_2 = 0 \text{ always results in } G \text{ being nonterminating.}$$

The crux of the simulation of $q_i \xrightarrow{C_2:=C_2-1} q_j$ lies in that if $b > 0$, then r_2 , together with w_1 , has to be applied in the first step. With a token in c'_2 (as a result of applying r_2), r'_2 can then be applied to ‘delay’ the execution of w_3 , since both are

of type W_2 . Notice that $s_{(i,j),2}^2$ is a place at which no ‘trap’ is defined. Hence, its outgoing transition w_3 can be delayed as long as it is needed. On the other hand, if $b = 0$ initially, then w_2 and w_3 must be applied at the same time in the second step, which eventually results in a token in f as our earlier discussion concludes.

• $p_i \xrightarrow{C_d=0} p_j$.

To this end, Figure 1(d) displays the PN structure used for the case $d = 2$. ($d = 1$ is similar.) Again, if the next transition to be simulated from p_j is ‘increment’, then $\varphi(v_2, p_j) = 2$.

It is clear that if v_1 is not applied immediately, π_1 forces a token to move from p_i to f under the 3-Max-Parallel mode. We now consider the case when $C_2 = 0$ and $C_1 > 0$. In this case, the firing sequence $(c_1)^a(p_i) \xrightarrow{(v_1,0,0)} (c_1)^a(t_{(i,j),2}^1) (t_{(i,j),2}^2) \xrightarrow{(v_2,v_3,0)} (c_1)^a(p_j)$ constitutes a legitimate sequence which deposits token(s) to place p_j to set up the simulation of the next transition. It is reasonably easy to see that all other transition sequences eventually move a token to f . Now suppose in the beginning of the simulation, $C_2 \neq 0$ (i.e., $b > 0$). Operating in the **3-Max-Parallel** mode forces transition r_2 to be taken immediately, along with the transition v_1 . Now in this case, places c_2' and $t_{(i,j),2}^2$ both contain tokens, forcing one of β_2 and γ_3 to be applied next (since both transitions r_2' and v_3 are of the same type W_2) – again causing f to be nonempty.

• $p_i \xrightarrow{C_d=C_d+1} p_j$

Figure 1(e) shows the PN structure designed for the case $d = 2$. ($d = 1$ is similar.) Again, if the next transition to be simulated from p_j is ‘increment’, then $\varphi(x_1, p_j) = 2$. Since in our design we always have two tokens in p_i prior the simulation of this transition, one must apply both x_1 and x_2 simultaneously; otherwise, δ_1 would have to be taken. The applications of x_1 and x_2 not only increment c_2 by one, but also inhibit r_1 and r_2 from been incorrectly executed.

Finally, a transition removing a token from p_k (the place corresponding to the accepting state of M) is included in T to force halting of G as soon as M reaches an accepting state. Upon halting, the value of l (i.e., the designated place) is m . Clearly, G does not halt if M never accepts, or during the course of the simulation, an illegitimate simulation step is carried out which moves a token to place f . This completes the proof of Statement (1) of the theorem. The proof of (2) can be carried out along a similar line as that of (1), and hence, is left to the reader. $\square$

In converting the PN constructed in the theorem above to a CS, the catalyst C_1 (resp. C_2) is needed to make sure that at most one transition in W_1 (resp. W_2) is simulated by the PN at each step. However, catalyst C_3 is not really needed in that we can convert each transition in W_3 to a rule of the form $a \rightarrow v$, i.e., a *non-cooperative rule* (without a catalyst). The reason is that in a legitimate simulation, at most two transitions (one from each W_i , $i = 1, 2$) are applied simultaneously. The **3-Max-Parallelism** requirement is simply to ensure that the simulation stays on the right track in order to have a chance to terminate. If an illegitimate step is taken, a type W_3 transition is forced to be applied, making the system nonterminating. Thus, the system can be constructed to have only *two* catalysts with catalytic

rules and noncooperative rules, and still be universal under the **3-Max-Parallel** mode. This was also shown in [8]. Notice that both the 3-catalyst system and the 2-catalyst system with catalytic and noncooperative rules are nondeterministic. In terms of the degree of parallelism, the difference between the two systems is the following. If the system operates in maximally parallel mode, along any computation path at most 3 rules are applied simultaneously for the former (because of the 3 catalysts), whereas for the latter, there might be some paths along which more than 3 rules are applied simultaneously (due to the presence of noncooperative rules). We are able to show (based on the construction in Figure 1) that at any point, no more than 4 noncooperative rules are applicable simultaneously in the 2-catalyst system with catalytic and noncooperative rules for simulating a 2-counter machine. This in turn implies that the degree of maximal parallelism along any computation path now becomes 6 (two catalysts plus 4 noncooperative rules) in such a system. Note, however, that imposing **3-Max-Parallelism** on such a system is sufficient to render the system universal. Note also that n , the dimension of the communication-free PN, which translates to the number of noncatalysts for the system, is also a function of the number of states, hence is unbounded.

We can improve the above results. We need the following lemma.

Lemma 1 *There exists a 2-counter machine U with counters C_1 and C_2 that is universal in the following sense. When U is given a description of an arbitrary 2-counter machine M as a positive integer in C_2 and an input m in C_1 , U accepts iff M with input m on its first counter and 0 on its other counter accepts.*

Proof. We sketch the construction of U . Let $L \subseteq \mathbb{N}$ be a recursively enumerable set of nonnegative integers accepted by a deterministic TM M with a *one-way* unary read-only input (with endmarkers) and one (two-way) read-write worktape, i.e., M when given n on its input tape (in unary with endmarkers), computes and accepts if n is in L .

It is well known that we can construct a universal deterministic TM U_1 with two read-only input tapes and one read-write tape that operates as follows. U_1 , when given n and a description x_M over the binary alphabet $\{1, 2\}$ of a deterministic TM M on its two input tapes, will simulate the computation of M on n and accepts iff M on input n accepts.

First, we convert U_1 to an equivalent universal deterministic TM U_2 , where the description of x_M is given as a unary string $Tally(x_M) = 1^{Num(x_M)}$, where $Num(x_M)$ is x_M interpreted as a number in 2-adic notation. The idea is for U_2 to read $Tally(x_M)$ on the input and convert it into 2-adic representation x_M on the first track of the worktape. Then U_2 uses x_M to simulate U on n using the second track of the worktape. So now U_2 has two read-only unary input tapes (with endmarkers) and a worktape.

Next, we convert the worktape of U_2 to a finite number of counters. Three counters are sufficient to simulate the TM worktape, as shown in [5, 6]. The new machine U_3 will now have two unary inputs and three counters.

Then we convert U_3 to a deterministic counter machine U_4 which will use two counters to simulate the two unary input tapes (note that the unary input tapes

are one-way) and three working counters, all of which are initially zero.

Thus, U_4 has 5 counters. Initially, the first counter contains n , the second counter contains $Tally(x_M)$, and the other 3 counters are zero.

Finally, we convert U_4 to a universal 2-counter machine U . Call the two counters of U C_1 and C_2 . Initially C_1 has value n and $C_2 = p_1^0 p_2^{Tally(x_M)} p_3^0 p_4^0 p_5^0$. Here $p_1, \dots, p_5$ are the first 5 prime numbers, and the values of the 5 counters of U_4 will be represented as the exponents of the prime numbers.

U first encode n (the initial value of C_1) into counter C_2 , so that at the end of the process, $C_1 = 0$ and C_2 contains $p_1^n p_2^{Tally(x_M)} p_3^0 p_4^0 p_5^0$. (Thus, the ‘exponent’ of p_1 will now represent the value of the first counter.) Then U simulates the computation of U_4 using only the two counters C_1 and C_2 and accepts iff U_4 accepts. The simulation is straightforward. $\square$

Therefore, we have:

Corollary 3 *There exists a fixed positive integer n such that if $L \subseteq \mathbf{N}$ is any recursively enumerable set of nonnegative integers, then L can be generated (accepted) by either a 1-membrane 3-Max-Parallel CS with 3 catalysts and n noncatalysts, or a 1-membrane 3-Max-Parallel P system with 2 catalysts and n noncatalysts with catalytic and noncooperative rules.*

2.5. 9-Max-Parallel 1-Membrane CS with One Catalyst

We now look at a model of a 1-membrane CS with only *one* catalyst C with initial configuration $C^k x$ for some string x of noncatalysts (thus, there are k copies of C). The rules allowed are of the form $Ca \rightarrow Cv$ or of the form $Caa \rightarrow Cv$, i.e., C catalyzes two copies of an object. Clearly the system operates in maximally parallel mode, but uses no more than k rules in any step. We call this system 1GCS. This system is equivalent to a restricted form of cooperative P system [15, 16]. A simple cooperative system (SCO) is a P system where the rules allowed are of the form $a \rightarrow v$ or of the form $aa \rightarrow v$, where a is a symbol and v is a (possibly null) string of symbols not containing a . Moreover, there is some fixed integer k such that the system operates in maximally parallel mode, but uses no more than k rule instances in any step. We can show the following:

Theorem 5 *1GCS (hence, also SCO) operating under the 9-Max-Parallel mode is universal.*

Proof. (sketch) To demonstrate universality, we show a subclass of PNs (which are computationally equivalent to 1GCS) to be capable of simulating arbitrary 2-counter machines under the **9-Max-Parallel** mode. Each of such PNs (P, T, φ) has the following restrictions: $\forall t \in T, |\{p \mid \varphi(p, t) > 0\}| = 1$, and $\forall p \in P, t \in T, (\varphi(p, t) \leq 2) \wedge \neg((\varphi(p, t) > 0) \wedge (\varphi(t, p) > 0))$. The equivalence between such PNs and 1GCS is obvious: transitions with $\varphi(p, t) = 1$ (resp., $= 2$) correspond to rules of the form $Ca \rightarrow Cv$ (resp., $Caa \rightarrow Cv$) in 1GCS. The **9-Max-Parallelism** semantics requires that at each step, a maximal subset of at most 9 distinct transitions from T be applied simultaneously.

Let M be a 2-counter machine with two counters C_1 and C_2 , and the set of states of M be $\{q_1, \dots, q_k\}$, where q_1 the initial state and q_k the final state. A

configuration of M is a triple (q, i, j) , where q is a state and i and j ($\in \mathbf{N}$) are the values of counters C_1 and C_2 , respectively. W.l.o.g., we assume that whenever M enters an accepting state, both counters are empty. In the remainder of the proof, we show how to construct a PN S with initial marking x in such a way that M accepts on initial counter values $C_1 = m$ and $C_2 = 0$ iff for some designated place a ,

1. (generator:) $m \in \{y(a) \mid y \in R_h(S, x)\}$,
2. (acceptor:) initial marking is $a^m y$ and $R_h(S, x) \neq \emptyset$, for some y not containing a .

The proof is somewhat involved. To allow the reader to have a better understanding for the intuition behind the proof, we give a high level description of how the simulation works before getting into the details. In the constructed PN graph, all the arcs are of weight one except otherwise annotated. Figure 2(a) illustrates a PN structure in which c_1 and c'_1 (resp., c_2 and c'_2) together hold the value of counter C_1 (resp., C_2) during the course of the simulation. The reason to use two places to simulate a counter (as opposed to using a single place for a counter as was done in the proof of Theorem 4) is that under the **9-Max-Parallel** mode, all (up to 9) the enabled transitions must be applied simultaneously, unless an enabled transition is in conflict with some other enabled transitions. In our setting, the key idea for ‘holding’ the value of a counter is to allow the tokens to circle around in a way illustrated in Figure 2(a). Also note that $\varphi(a_i, \alpha_i) = 2, i = 1, 2$.

Places appearing in Figure 2(a) serve for the following purposes:

- c_1, c'_1, c_2, c'_2 : if marking μ corresponds to configuration (q, i, j) of M during the course of the simulation, then $\mu(c_1) + \mu(c'_1) = i$, $\mu(c'_1) \leq 1$, $\mu(c_2) + \mu(c'_2) = j$, $\mu(c'_2) \leq 1$. In words, the number of tokens in c_1 and c'_1 (resp., c_2 and c'_2) together records the value of counter C_1 (resp., C_2).
- a_1 and a_2 : their presence or absence reflects the status of the counters, i.e., whether C_1 and C_2 , respectively, are empty or not. In any reachable marking μ , $\mu(a_1) \leq 2$ and $\mu(a_2) \leq 2$.
- f and f' play the same role as the ‘trap’ designed in the proof of Theorem 4. As such a trap is needed in several parts of the PN construction, we use Figure 2(b) as an abbreviation for the trap.

The simulation of a transition from states q to q' involves two *threads* evolving from the place representing q . See Figure 2(c). It should be noted that Figure 2(c) simply displays the simplified structure of a building block whose details depend on the transition to be simulated (increment, decrement, or test-for-zero) as well as on the token distribution among places in Figure 2(a). Starting with two tokens in q , a successful simulation moves two tokens along the two threads simultaneously and eventually deposits two tokens to q' to begin the simulation of the next transition. The counter values will also be updated or tested accordingly along this process.

Fig. 2. Building blocks of a PN under the **9-Max-Parallel** mode to simulate a 2-counter machine.

If an illegitimate step is taken in between, a token will eventually be forced to (under the **9-Max-Parallel** mode) move to the trap, making the computation nonterminating. More will be said about this as our proof unfolds. As mentioned earlier, a_1 (resp. a_2) serves as an indicator for telling whether counter C_1 (resp., C_2) is empty or not. It is critical in our design to never let the number of tokens in a_1 (and a_2) exceed 2 at all times. To achieve this, we simply synchronize the generations of a_1 and a_2 with the two threads at those places whose outgoing arcs are of weight 2. (Such places are called *synchronization places*.) As the frequency for a_1 to possess two tokens differs, the building block has to be tailored to fitting various (a finite number of) circumstances in the simulation. The choice for a proper module to carry out the next simulation is done nondeterministically, and an incorrect selection simply leads the simulation to nontermination with a token being deposited into f eventually.

In what follows, we show how the three types of transitions of a 2-counter machine can be simulated faithfully in our PN setting. For convenience, we use ' $\Rightarrow$ ' to denote one marking leading to another under the **9-Max-Parallel** mode.

(Increment counter C_1 : $q_i \xrightarrow{C_1 := C_1 + 1} q_j$)

As mentioned earlier, the PN module for simulating a transition depends not only on the transition itself, but also on the token distribution for the PN shown in Figure 2(a). Here we demonstrate the simulation for one case and from this case, it is reasonably easy to extend to the constructions of the remaining cases. The disparity between different cases mainly results from how early the tokens in a_1 and a_2 have to be synchronized in order to meet the constraints of $\mu(a_1) \leq 2$ and $\mu(a_2) \leq 2$, for every reachable marking μ during the course of the simulation.

Consider Figure 3(a) with the simulation starting from marking $(q_i)^2 (c_1)^m (c'_1) (c_2)^n (c'_2) (a_1)^2 (a_2)$, $m > 0$ and $n > 0$. The simulation is through the following sequence: $(q_i)^2 (c_1)^m (c'_1) (c_2)^n (c'_2) (a_1)^2 (a_2) \Rightarrow (v_1)^2 (v_2) (c_1)^{m+1} (c'_1) (c_2)^n (c'_2) (a_1) (a_2)^2 \Rightarrow (q_j)^2 (c_1)^{m+1} (c'_1) (c_2)^n (c'_2) (a_1)^2 (a_2)$. Notice that a_1 has to be

Fig. 3. Simulation of 'increment', 'decrement' and 'test-for-zero'.

synchronized right away since it contains two tokens initially. Place a_2 , however, contains a single token in the beginning; it need only be synchronized at the end of the simulation. At the end, two tokens are placed in q_j . The remaining cases are similar.

(Decrement counter C_1 : $q_i \xrightarrow{C_1 := C_1 - 1} q_j$)

Starting from marking $(q_i)^2 (c_1)^m (c'_1) (c_2)^n (c'_2) (a_1)^2 (a_2)$ with $m > 0$ and $n > 0$, consider Figure 3(b). (The value of counter C_1 is $m + 1$.) Clearly under the **9-Max-Parallel** mode, the following is a legitimate computation: $(q_i)^2 (c_1)^m (c'_1) (c_2)^n (c'_2) (a_1)^2 (a_2) \Rightarrow (u_1)^2 (u_2) (c_1)^m (c'_1) (c_2)^n (c'_2) (a_1) (a_2)^2 \Rightarrow (u_3)^2 (u_4)^2 (c_1)^{m-1} (c'_1) (c_2)^n (c'_2) (a_1)^2 (a_2) \Rightarrow (q_j)^2 (c_1)^{m-1} (c'_1) (c_2)^n (c'_2) (a_1) (a_2)^2$, and in the resulting marking, the value of counter C_1 (kept in c_1 and c'_1 together) becomes m . Note that in the above sequence, the decrement is done by removing a token from c'_1 . There are several other cases, which are all similar (in fact, simpler) to the above. The details are left to the reader.

Finally, we consider the simulation of a 'test-for-zero' transition.

(Test counter C_1 for zero: $q_i \xrightarrow{C_1 = 0} q_j$)

- (Counter $C_1 = 1$): Starting from marking $(q_i)^2 (c'_1) (c_2)^n (c'_2) (a_2)$, with $n > 0$, Figure 3(c) shows the PN module used for the simulation. Clearly under the **9-Max-Parallel** mode, $(q_i)^2 (c'_1) (c_2)^n (c'_2) (a_2) \Rightarrow (w_1) (w_2) (c_1) (c_2)^n (c'_2) (a_2)^2 \Rightarrow (w_3) (w_4)^2 (c'_1) (c_2)^n (c'_2) (a_1) (a_2) \Rightarrow (w_5) (w_6) (c_1) (c_2)^n (c'_2) (a_1) (a_2)^2 \Rightarrow (w_7) (w_8)^2 (c'_1) (c_2)^n (c'_2) (a_1)^2 (a_2) \Rightarrow (q_j)^2 (c_1) (c_2)^n (c'_2) (d) (a_2)^2$, for some d . Note that at the last step the two tokens in a_1 are forced to trigger a transition moving a token to d , which is either f (the trap) or a synchronization place in some other module. (The current simulation module cannot synchronize with a_1 as Figure 3(c) indicates.) Under **9-Max-Parallelism**, d (if not already f) is forced to deposit a token in f in

the next step. The key here is that if counter C_1 is not empty in the beginning, then after a certain number of steps two tokens will eventually be deposited in a_1 and since the thread corresponding to a_1 lacks any synchronization places (as Figure 3(c) indicates), the two tokens in a_1 cannot be consumed properly – causing the computation to be nonterminating. Note that the length along the thread associated with a_1 must be long enough so that a sufficient time is allowed to elapse for two tokens to be deposited in a_1 should there be a token in c_1 or c'_1 .

- (Counter $C_1 = 0$): Starting from marking $(q_i)^2(c_2)^n(c'_2)(a_2)$, with and $n > 0$, again see Figure 3(c). Clearly under the **9-Max-Parallel** mode, $(q_i)^2(c_2)^n(c'_2)(a_2) \Rightarrow (w_1)(w_2)(c_2)^n(c'_2)(a_2)^2 \Rightarrow (w_3)(w_4)^2(c_2)^n(c'_2)(a_2) \Rightarrow (w_5)(w_6)(c_2)^n(c'_2)(a_2)^2 \Rightarrow (w_7)(w_8)^2(c_2)^n(c'_2)(a_2) \Rightarrow (q_j)^2(c_2)^n(c'_2)(a_2)^2$.

The remaining cases are similar to the above.

For the acceptor case, a PN structure similar to the loop $y \rightarrow h_1 \rightarrow y' \rightarrow h_2 \rightarrow y$ in Figure 1(a) can be used in the initial phase of the simulation to generate an arbitrary number of tokens in c_1 . (Here c_1 is regarded as the designated place a .) Unlike the proof of Theorem 4, once c_1 becomes nonempty, its tokens keep spinning between c_1 and c'_1 ; in addition, tokens start to pile up in a_1 . To make sure that the number of tokens in a_1 never exceeds two, a proper synchronization (similar to what we have seen thus far for the simulation of the three types of transitions) has to be carried out in the process of putting tokens in c_1 . For the generator case, the idea is that upon reaching the final state of M , the PN starts moving one token at a time from c_1 (using a mechanism similar to the ‘decrement’ case) to the designated place a from which the output number is observed.

To complete the proof of this theorem, consider the degree of max-parallelism required in our simulation. The structure in Figure 2(a) requires maximally 6 transitions (namely, r_1, r'_1, r_2, r'_2 and two more to remove tokens from a_1 and a_2) to be fired simultaneously. In addition, the two threads in Figure 2(b) need two transitions to be applied in parallel. Finally, an extra degree of parallelism is needed to ensure that if an illegitimate step is taken in the simulation, a token is forced to be put in f . Hence, under the **9-Max-Parallel** mode the only way for the simulation to halt eventually is for M to halt, and every step of the simulation is legitimate. $\square$

3. Sequential 1-Membrane Communicating P Systems

Consider the model of a communicating P system (CPS) with only *one* membrane, called the skin membrane [19]. The rules are of the form: (1) $a \rightarrow a_x$, (2) $ab \rightarrow a_x b_y$, (3) $ab \rightarrow a_x b_y c_{come}$, where a, b, c are objects, x, y (which indicate the directions of movements of a and b) can only be *here* (i.e., the object remains in the membrane) or *out* (i.e., the object is expelled into the environment). The third rule brings in an object c from the environment into the skin membrane. In the sequel, we omit the designation *here*, so that objects that remain in the membrane will not have this subscript. There is a fixed finite set of rules in the membrane. At the beginning, there is a fixed configuration of objects in the membrane.

Assume that the computation is *sequential*; i.e., at each step there is only one

application of a rule (to one instance). So, e.g., if nondeterministically a rule like $ab \rightarrow a_{here}b_{out}c_{come}$ is chosen, then there must be at least one a and one b in the membrane. After the step, a remains in the membrane, b is thrown out of the membrane, and c comes into the membrane. There may be several a 's and b 's, but only one application of the rule is applied. Thus, there is *no* parallelism involved. The computation halts when there is no applicable rule. We are interested in the multiplicities of the objects when the system halts.

One can show that a 1-membrane CPS can be simulated by a vector addition system (VAS) (this is a special case of a theorem in the next section). However, the converse is not true – it was shown in [3] that a sequential 1-membrane CPS can only define a semilinear set.

4. Sequential 1-Membrane Extended CPS (ECPS)

Recall that 1-membrane CPS's operating sequentially define only semilinear sets. In contrast, we shall see in the next section that sequential 2-membrane CPS's are equivalent to VASS.

There is an interesting generalization of a 1-membrane CPS, we call extended CPS (or ECPS) – we add a fourth type of rule of the form: $ab \rightarrow a_x b_y c_{come} d_{come}$. That is, two symbols can be imported from the environment. We shall see below that ECPS's are equivalent to VASS's.

Let G be an n -dimensional VASS. Clearly, by adding new states, we may assume that all transitions in G have the form: $p_i \rightarrow (p_j, +1_h)$, $p_i \rightarrow (p_j, -1_h)$. The above is a short-hand notation. The $+1_h$ is addition of 1 to the h -th coordinate, and -1_h is subtraction of 1 from the h -th coordinate. All other coordinates are unchanged. Note at each step, the state uniquely determines whether it is a '+1 transition' or a '-1 transition'.

For constructing the ECPS S equivalent to G , we associate symbol p_i for every state of the VASS, a_h for every coordinate (i.e., position) h in the transition. We also define a new special symbol c . So the ECPS has symbols $p_1, \dots, p_s$ (s is the number of states), $a_1, \dots, a_n$ (n is dimension of the VASS), and c .

Then a transition of the form $p_i \rightarrow (p_j, +1_h)$ in G is simulated by the following rule in S : $p_i c \rightarrow p_{i(out)} c_{here} p_{j(coming)} a_{h(coming)}$

A transition of the form $p_i \rightarrow (p_j, -1_h)$ in G is simulated by the following rule in S : $p_i a_h \rightarrow p_{i(out)} a_{h(out)} p_{j(coming)}$

If the VASS G has starting point $\langle p_1, v \rangle$, where $v = (i_1, \dots, i_n)$ and p_1 is the start state, then ECPS S starts with the word $p_1 a_1^{i_1} \dots a_n^{i_n} c$. Clearly, S simulates G .

Conversely, suppose we are given an ECPS S over symbols $a_1, \dots, a_n$ with initial configuration w and rules $R_1, \dots, R_k$. The VASS G has states $R_0, R_1, R'_1, \dots, R_k, R'_k$ and starting point $\langle R_0, v_0 \rangle$, where v_0 is the n -dimensional vector in $\mathbb{N}^n$ representing the multiplicities of the symbols in the initial configuration w . The transitions of G are defined as follows:

1. $R_0 \rightarrow (R_i, zero)$ for every $1 \leq i \leq k$ is a transition, where *zero* represents the

zero vector.

2. If R_i is a rule of the form $a_h \rightarrow a_{hx}$, then the following are transitions:

$$R_i \rightarrow (R'_i, -1_h)$$

$$R'_i \rightarrow (R_j, d_{hx}) \text{ for every } 1 \leq j \leq k, \text{ where } d_{hx} = 0_h \text{ if } x = (\text{out}) \text{ and } d_{hx} = +1_h \text{ if } x = (\text{here}).$$

(As before, $-1_h, 0_h, +1_h$ mean subtract 1, add 0, add 1 to h , respectively; all other coordinates are unchanged.)

3. If R_i is a rule of the form $a_h a_r \rightarrow a_{hx} a_{ry}$, then the following are transitions:

$$R_i \rightarrow (R'_i, -1_h, -1_r)$$

$$R'_i \rightarrow (R_j, d_{hx}, d_{ry}) \text{ for every } 1 \leq j \leq k, \text{ where } d_{hx} \text{ and } d_{ry} \text{ are as defined above.}$$

(Note that if $h = r$, then $(R'_i, -1_h, -1_r)$ means $(R'_i, -2_h)$, i.e., subtract 2 from coordinate h .)

4. If R_i is a rule of the form $a_h a_r \rightarrow a_{hx} a_{ry} a_{s(\text{come})}$, then the following are transitions:

$$R_i \rightarrow (R'_i, -1_h, -1_r)$$

$$R'_i \rightarrow (R_j, d_{hx}, d_{ry}, +1_s) \text{ for every } 1 \leq j \leq k, \text{ where } d_{hx} \text{ and } d_{ry} \text{ are as defined above.}$$

5. If R_i is a rule of the form $a_h a_r \rightarrow a_{hx} a_{ry} a_{s(\text{come})} a_{t(\text{come})}$, then the following are transitions:

$$R_i \rightarrow (R'_i, -1_h, -1_r)$$

$$R'_i \rightarrow (R_j, d_{hx}, d_{ry}, +1_s, +1_t) \text{ for every } 1 \leq j \leq k, \text{ where } d_{hx} \text{ and } d_{ry} \text{ are as defined above.}$$

It follows from the construction above that G simulates S . Thus, we have:

Theorem 6 *Sequential 1-membrane ECPS and VASS are equivalent.*

We can generalize rules of an ECPS further as follows: (1) $a_{i_1} \dots a_{i_h} \rightarrow a_{i_1 x_1} \dots a_{i_1 x_h}$ and (2) $a_{i_1} \dots a_{i_h} \rightarrow a_{i_1 x_1} \dots a_{i_1 x_h} c_{j_1 \text{come}} \dots c_{j_l \text{come}}$ where $h, l \geq 1$, and $x_m \in \{\text{here}, \text{out}\}$ for $1 \leq m \leq h$, and the a 's and c 's are symbols. Call this system ECPS+. Generalizing the constructions in the proof of Theorem 6, we can show ECPS+ is still equivalent to a VASS. Thus, we have:

Corollary 4 *The following systems are equivalent: Sequential 1-membrane ECPS, sequential 1-membrane ECPS+, and VASS.*

Using rules of types of 1 and 2 above, we can define the three versions of parallelism as in Section 2.3, and we can prove the following result. Note that the first part was shown in [9].

Theorem 7 *For $n = 2$, a 1-membrane CPS (and, hence, also 1-membrane ECPS+) operating under the n -Max-Parallel mode can define a recursively enumerable set. For any n , a 1-membrane ECPS+ operating under $\leq n$ -Parallel mode or n -Parallel mode is equivalent to a VASS.*

5. Multi-Membrane CPS and ECPS

In this section, we look at CPS and ECPS with multiple membranes. Now the subscripts x, y in the CPS rules $a \rightarrow a_x$, $ab \rightarrow a_x b_y$, $ab \rightarrow a_x b_y c_{come}$ (and $ab \rightarrow a_x b_y c_{come} d_{come}$ in ECPS) can be *here*, *out*, or *in_j*. As before, *here* means that the object remains in the membrane containing it, *out* means that the object is transported to the membrane directly enclosing the membrane that contains the object (or to the environment if the object is in the skin membrane), and *come* can only occur within the outermost region (i.e., skin membrane). The designation *in_j* means that the object is moved into a membrane, labeled j , that is directly enclosed by the membrane that contains the object.

In Section 3, we saw that a sequential 1-membrane CPS can only define a semi-linear set. We can show that if the system has two membranes, it can simulate a vector addition system. Thus, we have:

Theorem 8 *A sequential 2-membrane CPS S can simulate a VASS G .*

In Theorem 6, we saw that a sequential 1-membrane ECPS can be simulated by a VASS. The construction can be extended to multi-membrane ECPS. Recall that we now allow rules of the form: $ab \rightarrow a_x b_y c_{come} d_{com}$.

Suppose that S has m membranes. Let $\{a_1, \dots, a_n\}$ be the set of symbols (objects) that can occur in the configurations of S . Then each reachable configuration of S is an mn -tuple $(v_1, \dots, v_m)$, where v_q is an n -tuple representing the multiplicities of the symbols $a_1, \dots, a_n$ in membrane q . Then the set of all reachable configurations of S is a subset of $\mathbf{N}^{mn}$. Let $R_1, \dots, R_k$ be the rules in S . Note that R_i not only gives the rule but also the membrane where it appears. The construction of the VASS G simulating S is similar to the construction described in the second part of the proof of Theorem 6. In fact the construction also works for ECPS+. Since a sequential 2-membrane CPS can simulate a VASS, we have:

Theorem 9 *The following are equivalent: VASS, sequential 2-membrane CPS, sequential 1-membrane ECPS, sequential multi-membrane ECPS, and sequential multi-membrane ECPS+.*

Finally, we observe that Theorem 7 extends to multi-membrane CPS:

Theorem 10 *For any n , a multi-membrane ECPS+ operating under $\leq n$ -Parallel mode or n -Parallel mode is equivalent to a VASS.*

6. Conclusion

We showed in this paper that P systems that compute in sequential or limited parallel mode are strictly weaker than systems that operate with maximal parallelism for two classes of systems: multi-membrane catalytic systems and multi-membrane communicating P systems. Our proof techniques can be used to show that many of the P systems that have been studied in the literature (including ones with membrane dissolving rules) operating under sequential or limited parallelism with unprioritized rules can be simulated by vector addition systems.

References

1. H. G. Baker. Rabin's proof of the undecidability of the reachability set inclusion problem for vector addition systems. In *C.S.C. Memo 79, Project MAC, MIT*, 1973.
2. E. Csuhaj-Varju, J. Dassow, J. Kelemen, and Gh. Paun. *Grammar Systems: A Grammatical Approach to Distribution and Cooperation*. Gordon and Breach Science Publishers, Inc., 1994.
3. Z. Dang and O. H. Ibarra. On P systems operating in sequential and limited parallel modes. In *Pre-Proc. 6th Workshop on Descriptive Complexity of Formal Systems*, 2004.
4. J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. In *Proc. Fundamentals of Computer Theory*, volume 965 of *Lecture Notes in Computer Science*, pages 221–232. Springer, 1995.
5. P. C. Fischer. Turing machines with restricted memory access. *Information and Control*, 9:364–379, 1966.
6. P. C. Fischer, A. R. Meyer, and A. L. Rosenberg. Counter machines and counter languages. *Mathematical Systems Theory*, 2:265–283, 1968.
7. R. Freund. Sequential P-systems. Available at <http://psystems.disco.unimib.it>, 2000.
8. R. Freund, L. Kari, M. Oswald, and P. Sosik. Computationally universal P systems without priorities: two catalysts are sufficient. Available at <http://psystems.disco.unimib.it>, 2003.
9. R. Freund and A. Paun. Membrane systems with symport/antiport rules: universality results. In *Proc. WMC-CdeA2002*, volume 2597 of *Lecture Notes in Computer Science*, pages 270–287. Springer, 2003.
10. M. H. Hack. The equality problem for vector addition systems is undecidable. In *C.S.C. Memo 121, Project MAC, MIT*, 1975.
11. J. Hopcroft and J.-J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8(2):135–159, 1979.
12. D.T. Huynh. Commutative grammars: The complexity of uniform word problems. *Information and Control*, 57:21–39, 1983.
13. O. Ibarra, Z. Dang, and O. Egecioğlu. Catalytic P systems, semilinear sets, and vector addition systems. *Theoretical Computer Science*, 312(2-3): 379–399, 2004.
14. E. Mayr. Persistence of vector replacement systems is decidable. *Acta Informat.*, 15:309–318, 1981.
15. Gh. Paun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
16. Gh. Paun. *Membrane Computing: An Introduction*. Springer-Verlag, 2002.
17. Gh. Paun and G. Rozenberg. A guide to membrane computing. *Theoretical Computer Science*, 287(1):73–100, 2002.
18. J. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall, Englewood Cliffs, NJ, 1981.
19. P. Sosik. P systems versus register machines: two universality proofs. In *Pre-Proceedings of Workshop on Membrane Computing (WMC-CdeA2002)*, Curtea de Arges, Romania, pages 371–382, 2002.
20. J. van Leeuwen. A partial solution to the reachability problem for vector addition systems. In *Proceedings of STOC'74*, pages 303–309.

Copyright of International Journal of Foundations of Computer Science is the property of World Scientific Publishing Company and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.